DOCUMENT RESUME

ED 078 681 EM 011 237

AUTHOR	Koffman, Elliot B.; And Others
TITLE	An Intelligent CAI Monitor and Generative Tutor. Interim Report.
INSTITUTION	Connecticut Univ., Storrs. Dept. of Electrical Engineering.
SPONS AGENCY	National Inst. of Education (DHEW), Washington, D.C.
REPORT NO	P-020193
PUB DATE	May 73
NOTE	67p.; See Also EM 011 176 and EM 011 177
EDRS PRICE	MF-$0.65 HC-$3.29
DESCRIPTORS	Algebra; Branching; College Students; *Computer Assisted Instruction; Computers; *Design; Engineering Education; High School Students; Individual Instruction; Laboratory Procedures; *Problem Solving; Program Descriptions; Programed Tutoring; Programing; *Tutorial Programs; Tutoring
IDENTIFIERS	CAILD; COMSEQ; Generative Tutor; MALT

ABSTRACT

Design techniques for generative computer-assisted-instructional (CAI) systems are described in this report. These are systems capable of generating problems for students and of deriving and monitoring solutions; problem difficulty, instructional pace, and depth of monitoring are all individually tailored and parts of the solution algorithms can be used to analyze incorrect student responses and to direct remediation. A generative CAI system which teaches logic design and machine-language programing is discussed. This system covers material for an introductory electrical engineering course and is intended to supplement regular instruction by providing practice in problem solving. Also described is a companion system for teaching laboratory principles in which students learn to construct combinational or sequential logic circuits using standard integrated circuits. The student's logic circuit is automatically interfaced to the computer for testing and the computer aids in debugging the circuit. Work in progress on the design of a tutor for high school algebra is also related. Finally, a formal mathematical approach to problem generation and solution is presented. (Author)

Interim Report

Project No. 020193
Grant No. OEG-0-72-0895

Elliot B. Koffman
Sumner E. Blount
Thomas Gilkey
James Perry
Martin Wei
University of Connecticut
Department of Electrical Engineering
Storrs, Connecticut 06268

An Intelligent CAI Monitor and Generative Tutor

May 1973

National Institute of Education

ABSTRACT

This paper describes design techniques for generative computer-assisted instruction (CAI) systems. These are systems which are capable of generating problems for students and deriving and monitoring the solutions to these problems. The difficulty of the problem, the pace of instruction, and the depth of monitoring are all tailored to the individual student. Parts of the solution algorithms can also be used to analyze an incorrect student response and determine the exact nature of the student's error in order to supply him with meaningful remedial comments.

A generative CAI system which teaches logic design and machine-language programming will be discussed. This CAI system covers the material in an introductory course in digital system aimed at electrical engineering juniors. It does not replace classroom lectures or the textbook, but instead serves to provide practice and instruction in applying this material to solve problems.

In addition, a companion system to teach laboratory principles has been designed. This system teaches a student how to construct a combinational or sequential logic circuit using standard integrated circuits. The student's logic circuit is automatically interfaced to the computer and tested; the computer then aids the student in debugging his circuit.

Work in progress on the design of a tutor for high-school algebra, which teaches students how to solve algebra word problems, is also described. Finally, a formal mathematical approach to problem generation and solution is presented.

Interim Report

Project No. 020193
Grant No. OEG-0-72-0895

An Intelligent CAI Montior and Generative Tutor

Elliot B. Koffman
Sumner E. Blount
Thomas Gilkey
James Perry
Martin Wei

University of Connecticut
Department of Electrical Engineering
Storrs, Connecticut 06268

May 1973

The research reported herein was performed pursuant to a grant with the National Institute of Education, U. S. Department of Health, Education and Welfare. Contractors undertaking such projects under Government sponsorship are encouraged to express freely their professional judgement in the conduct of the project. Points of view or opinions stated do not, therefore, necessarily represent official National Institute of Education position or policy.

U. S. Department of
Health, Education and Welfare
National Institute of Education

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Over the past two years a set of three computer-assisted instruction (CAI) systems has been designed around an introductory course in computer science. This course is taught in the Electrical Engineering Department at the University of Connecticut and is required of all Electrical Engineering students. It is also taken by students in other departments who wish to minor in computer science.

The first system, COMSEQ, (Koffman, 1972) introduces students to the design and simplification of COMbinational and SEQuential digital logic circuits. It starts off by teaching the binary, octal, and hexadecimal number systems and logical and arithmetic operations within these systems. It covers the design of combinational circuits and the Karnaugh Map and Quine-McKluskey methods of minimization. Finally, it teaches how to use Flip-Flops in the analysis and synthesis of sequential logic circuits.

The second CAI system, MALT, (MAchine Language Teacher - Blount and Koffman, 1972) teaches students how to program a minicomputer using machine language. The computer used is similar to the Digital Equipment Corporation PDP-8. For simplicity, the MALT version has only 377 (octal) words of core.

The third system, CAILD, (Computer Aided Instruction in Logic Debugging-Wei, 1973) instructs students in the design and debugging of digital logic circuits. The circuits are built by the student on a special logic board which contains TTL integrated circuits. These include NAND gates, inverters, and JK and D Flip-Flops. Sequential circuits with two external inputs and outputs and up to sixteen states can be constructed. Combinational circuits with up to six inputs and sixteen outputs are also possible.

These systems are oriented towards teaching problem solving and assist the student in learning how to apply the concepts introduced in class or the textbook (Booth, 1971). The first two systems have been used extensively and appear to serve very well as a replacement for coventional homework. They have the advantage of easing a beginning student into the problem and providing prompt and pertinent remedial feedback when he goes astray. Both systems are programmed on the IBM 360/65 and are available through the CPS (Conversational Programming System) time-sharing system. Students can call these programs from any of the forty CPS terminals on campus and continue with their CAI work whenever they wish.

Both systems are fully "generative". This means that the students do not work "canned" or pre-stored problems. The system generates, for each student, problems which are individualized with respect to his previous performance in the CAI course. The problems generated are normally not too difficult nor too easy and are different from other problems attempted. The generation process does not consist of merely plugging randomly generated parameters into preset question formats; rather, the problem format itself is often constructed from a set of basic problem elements. This provides for a wide variety of problems.

Since the actual problem presented is not predetermined, neither is the solution. COMSEQ and MALT both derive the solution to the problem along with the students. The speed at which the student progresses through the solution and the frequency with which his solution is monitored depend on his previous performance. Beginning students are led by the hand; whereas, COMSEQ and MALT will provide portions of the solution for advanced students and free them

to concentrate on the more complex aspects of the problem.

The third system, CAILD is capable of degugging any circuit which the student has designed. He must specify the logical equations which describe his circuit in Sum-of-Product form. The student logic board is automatically interfaced to the Digital Equipment Corporation PDP-9 computer which then controls the debugging process.

The emphasis is on teaching good debugging procedures rather than automating fault detection. Circuit inputs and Flip-Flop state values are controlled by CAILD. Critical output logic values are monitored. In addition, the student is directed to utilize a test-probe to monitor suspicious points in his circuit much as he would use an oscilloscope probe. CAILD compares this information with its model of the student's circuit to help him trace an error to its source. CAILD also allows a student to test his circuit after it has been debugged by applying current input and state conditions and displaying the next state and output values.

Taken together, these three CAI systems comprise an integrated package of programs which cover most aspects of logic design and machine language programming. The intent is not so much to instruct a student in the basic concepts but to guide him in applying what he has learned to solve problems, write programs, or debug and test circuits.

Chapters two through five describe these systems and their application in the course. Chapter six describes a tutor for high school algebra whose design is strongly influenced by COMSEQ. The classes of problems which the system can handle are word problems, manipulation problems, and graphing problems. Word problems test a student's reasoning and problem-solving

ability; manipulation problems test his grasp of the fundamental skills and techniques; graphing problems test his basic understanding of graphic methods and ability to plot equations.

Chapter seven is a formal model for problem generation and solution. Problem generation is described as originating from a semantic network (Quillian, 1969). This process is an extension of Carbonell's work to quantitative problem solving (Carbonell, 1970) and is motivated by Polya's classic work on problems (Polya, 1945). Indeed, this research can be thought of as a step in the automation of Polya's heuristics. Problem solution is closely related to problem generation. The plan for problem solution is generated at the same time that the problem is generated.

II. The COMSEQ System

A. Overview

The system is designed to be extremely flexible in that it can completely control the progress of a student through the course, selecting concepts for study on an individual basis and generating problems. Alternatively, the student can assume the initiative and determine his own areas of study and/or supply his own problems.

In addition, the system also operates in a "problem-solver" mode. In this mode, the student specifies the concept area and his problem, and the system will crank out the solution without further interaction. It is anticipated that students in later courses and the digital laboratory will utilize this mode for solving complex minimization problems and determining the relative merits of different state assignments.

Figure 1 is a block diagram of the system functions. Subsequent sections of this paper will describe how these functions are accomplished. As has been mentioned, the student can assume the initiative and by pass the Concept Selector and/or Problem Generator (indicated by dashed lines from Student box). The bottom part of Figure 1 shows the student exercising full control in the problem solver mode.

B. Concept Selector

When the system is in control of the interaction, it attempts to individualize the depth and pace of instruction presented to each student. A model of each student is kept which summarizes his past performance. In each of the course concepts. Table 1 shows the contents o a student record.

A student's LEVEL is a real number between 0 and 3. The last level change is the total change during the last use of the concept to solve a

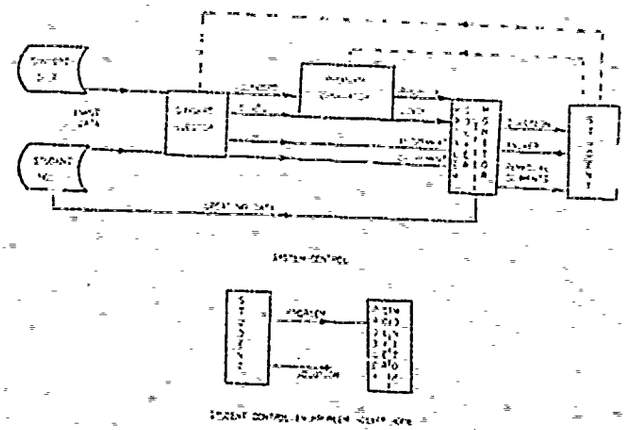


Figure 1

System Block Diagram

TABLE I—Student Record

Student Name E B Koffman Master Ave. 1.8 Current Plateau 5

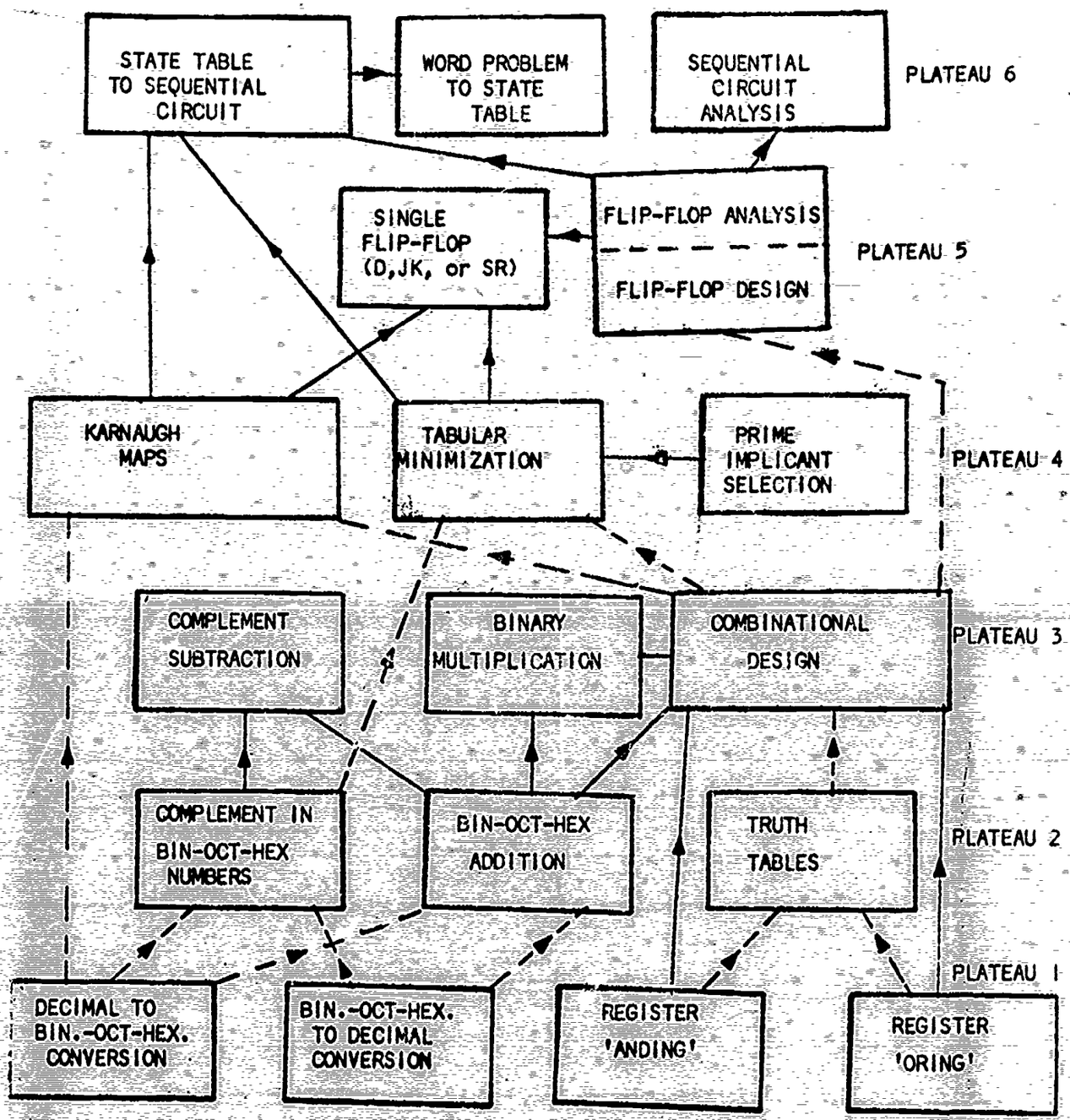
	Concept #1	Concept #2	. . .	Concept #30
Level	2.2	1.5		0.5
Last level change	.1	.5		−.2
Weighted Av. level change	.5	.6		−.1
Date of last call	3/15	3/17		4/20
Sequential order of last call	25	31		48
No. of times called in 0-1 range	2	2		3
No. of times called in 1-2 range	2	1		0
No. of times called in 2-3 range	0	0		0
No. of problems generated	2	3	. . .	3

problem. A small increase in LEVEL (.07 to .28) occurs for each correct answer and a small decrease for each incorrect answer. The student's LEVEL in a concept determines the difficulty of the problem generated and the amount of interaction he receives during its solution. In calculating the weighted average level change, the most recent change in a concept's LEVEL has the greatest effect. Normally, the number of problems generated in a concept area for a student who is performing well is around three or four.

In addition, the system is supplied with a concept tree which indicates the degree of complexity (plateau) of a concept and its relationship with other concepts in the course. A sample tree for the concepts currently covered is shown in Figure 2. This represents the author's interpretation of the relationship between course concepts. There are alternate interpretations which are just as valid.

The system uses each student's record to determine how quickly he should progress through the tree of concepts, the particular path which should be followed, the degree of difficulty of the problem to be generated, and the depth of monitoring and explanation of the problem solution.

Since there are a large number of concepts available for study, the system attempts to select the next concept in such a way as to make optimal use of the student's time. The goal is to pace the student through the concepts quickly enough so that he does not become bored or unmotivated and yet not so fast that he becomes unduly confused.



END: A → B CONCEPT A IS A PREREQUISITE OF CONCEPT B.

A → B CONCEPT A IS A PREREQUISITE OF CONCEPT B; CONCEPT A MAY BE USED AS A SUB-CONCEPT BY B.

Note: The relation "is a prerequisite of" is transitive (A is a prerequisite of B, B is a prerequisite of C, implies A is a prerequisite of C)

FIGURE 2 CONCEPT TREE

Each student is assigned a master average when he first logs onto the CAI system. This could be a function of his I.Q. or class standing (In the past, each student has been arbitrarily assigned an intial master average of 2). This value changes as the system gains experience with a student.

A student's master average controls the speed with which he jumps from one plateau of the concept tree to the next. In order to jump to the next higher plateau, the average of his levels of achievement in all concepts at and below the current plateau must exceed his master average. Consequently, the lower a student's master average, the faster he will progress. Each student's master average is updated after the completion of a problem.

Once the student's plateau has been determined, the system selects a set of candidate concepts from this plateau and those below it if necessary. In order to qualify as a candidate concept, the average of the student's levels of achievememt in all prerequisites for this concept (as determined for the concept tree) must exceed his master average. If this is not the case, the prerequisite concept in which the student has the lowest level is selected as a candidate in its place. This provides for automatic review of selected concepts at lowe plateaus.

The system then chooses one concept from among the candidates. Each concept is evaluated based on a number of factors such as the time elapsed since its last use, the "stability" of its current level, the sign and magnitude of its most recent level change (negative changes are weighted more heavily), and its relevance to other concepts as determined by the number of branches of the tree connected to it. Each of these factors is multiplied by a weight.

The highest scoring concept is selected for presentation to the student. The student always has the option of vetoing this selection and choosing his own concept or accepting the system's second best choice.

If the course-author is not satisfied with the manner in which the system is selecting concepts, it is relatively easy to modify the concept tree or change the weights associated with each of the factors affecting concept selection. The concept selector is completely course-independent. Consequently, it can be used in conjunction with another course if it is provided with a new concept tree.

It is also quite easy to add new concepts to a course. A solution algorithm and problem generator must be programmed for each new concept. A calling sequence is established which ensures that the problem generator passes the solution algorithm all necessary problem parameters. This sequence is appended to the list of concept calling sequences already available. The next step is to enter the author-mode of the generative CAI system in which the course-author or instructor is assisted in making the required additions to the concept-tree. The instructor can also examine student performance records while in the author-mode and see how his class is progressing.

Table 2 gives an example of the interaction between COMSEQ and a student. The first letter of each student response is underlined. The student is attempting to determine the combinational logic required to build a sequential circuit for the state table shown.

TABLE 2

Sample Student-System Interaction

Derive the truth table for the sequential network
which yields the following state table.

CURRENT STATE	INPUT-X1 X1=0	INPUT-X1 X1=1
	NEXT STATE / OUTPUT	
Q0	Q2/0	Q1/0
Q1	Q1/0	Q0/0
Q2	Q2/0	Q0/0
Q3	Q0/0	Q2/0

Use SR Flip Flops.

Is this problem ok?

yes

How many Flip-Flops are needed?

3

No. Since, there are 4 states, 2 Flip-Flops are needed.
3 Flip-Flops would be used for 5 to 8 states.
The following is the state assignment:
Each state will assume its normal binary value

STATE	Y1Y2
Q 0	00
Q 1	01
Q 2	10
Q 3	11

The table for describing the required combinational logic follows:

		CURRENT	NEXT				
	INPUT	STATE	STATE	FF1	FF2		OUTPUT
ROW	X1	Y1Y2	Y1Y2	S1 R1	S2 R2		
0	0	00					

What is the next state for row 0?

Q1

No. The current state corresponds to Q0.

TABLE 2 continued

The next state for input X1 = 0 is Q2

This corresponds to Y1Y2 = 10

What is the value of the output?

0

Type in S1

1

Type in R1

0

Type in S2

0

Type in R2

1

Your answer is ok but a better answer is S2 = 0, R2 = D (D is a 'don't care')

ROW	X1	Y1Y2	Y1Y2	S1	R1	S2	R2	OUTPUT
0	0	00	10	1	0	0	D	0
1	0	01						

What is the next state for row 1 ?

.

.

.

Note: The student continues building the truth table for S1, R1, S2, R2.

After he finishes, he can continue with CAI and minimize the combinational logic if he wishes.

C. Problem Generation and Solution

As can be seen from the concept tree, there are nineteen distinct concepts covered by COMSEQ. Associated with each concept is a problem generator and a solution algorithm. The problem generation routines are very much interrelated in that they share several common subroutines. The subroutines are of two basic types. The first type is driven by a probabilistic decision table. All of the decisions which affect problem complexity are identified and comprise the rows of the table. There are nine columns as there are at most three possible outcomes for each decision in each of the three LEVEL ranges (0-1, 1-2, 2-3). In Table 3, the entry in row 2 column 8 means that the probability of a state-table problem having one input bit is .2 for a student with LEVEL between 2 and 3.

The second type of problem generator is driven by a probabilistic grammar. A probabilistic grammar is a formal language in which each rewrite rule is assigned a probability of being applied. For several problem types, the generation process is controlled by both decision tables and probabilistic grammars.

Table 4 is an example of a probabilistic grammar which generates logical expressions for use in the concepts dealing with truth tables and sequential circuit analysis. Each of the symbols to the left of the arrow is a non-terminal symbol and can be replaced by the string of symbols to the right of the arrow. The incomplete logical expression is scanned from left to right for non-terminal symbols (A,*). When the non-terminal symbol A is found, $(P_a + P_b)$ represents the probability of increasing the length of the expression where:

$$P_a(t) = P_b(t) = .75C/(n(t) + 1.5C-1.5) \qquad (1)$$

TABLE 3

Decision Table for Generating State Table Problems

		Level Range								
		0-1			1-2			2-3		
Number of states(2|4|5-8)	1	.7	.3	0	0	.5	.5	0	0	1
Number of input bits (0|1|2)	2	.5	.5	0	0	.5	.5	.5	.2	.8
Output (absent|present|-)	3	1	0	-	.5	.5	-	.2	.8	-
	Row / Column	1	2	3	4	5	6	7	8	9

TABLE 4

Grammar for Logical Expressions

Probability: Rewrite rule

P_a: A→(A*A)

P_b: A→(¬A)

P_{c1}: A→P

P_{c2}: A→Q

P_{c3}: A→R

P_{c4}: A→S

P_d: *→∨(or)

P_d: *→∧(and)

P_e: *→↑(NAND)

P_e: *→↓(NOR)

P_e: *→⊕(exclusive or)

Constraints $0 \leq P_a, P_b, P_{ci}, P_d, P_e \leq 1$

$P_a + P_b + \sum_i P_{ci} = 1$

$2P_d + 3P_e = 1$

where n(t) is the current length and C is the number of variables in the expression ($C \geq 2$).

Since $(P_a + P_b)$ is inversely proportional to the current length, the logical expressions do not become unwieldy. If the random number generated indicates that the expression should not be extended, one of the terminal symbols {P, Q, R, S} replaces A.

If the non-terminal symbol is *, one of the five logical operators {V, Λ, ↑, ↓, ⊕} replaces it. P_e increases with LEVEL, while P_d decreases. Hence, the more difficult operators are more likely in expressions generated for advanced students.

Once the problem has been generated, all pertinent parameters are passed to the solution algorithm. As each sub-task of the solution is completed, a decision is made whether or not to question the student on this part of the solution. Students with LEVEL < 1 in the concept will, of course, be asked the most questions. The student's LEVEL and, hence, the depth of monitoring is dynamically changed as the problem continues. Each correct answer causes an increase in LEVEL ($.07 \leq |\Delta L| \leq .28$) while each incorrect answer causes a decreases in LEVEL.

D. Error Analysis

When a student is incorrect, the system analyzes his response to determine the cause of his error and provides pertiment remedial feedback. The concept which teaches the use of Karnaugh Maps is a good example of what can be done in the way of error analysis. In the sample problem shown in Table 5, the student is attempting to identify all squares with truth value 1 which must be picked up in pairs ("2-SQUARES"). The only required pair consists of minterms 8,9. The output shows the remedial response given to a

TABLE 5

ANSWER ANSLYSIS IN KARNAUGH MAPS

PROBLEM: Minimize a function which has 5, 7, 8, 9, 13, 15 as minterms using the Karnaugh Map method.

			1
	1	1	1
	1	1	

KARNAUGH MAP

0	4	12	8
1	5	13	9
3	7	15	11
2	6	14	10

NUMBERING SYSTEM FOR MINTERMS

Part of Student Solution

```
PLEASE ENTER THE 2-SQUARES.
WHEN YOU ARE DONE, TYPE 99.
5,4
4 DOES NOT HAVE TRUTH VALUE 1.
5, 7
THESE TERMS ARE PART OF A 4-SQUARE.    (A)
5, 15
THESE TERMS ARE NOT ADJACENT           (B)
13, 9
13,9 IS A 2-SQUARE, BUT IT SHOULD NOT  (C)
BE PICKED UP YET.
8,9
CORRECT.  ENTER MORE 2-SQUARES OR 99.
99
PLEASE ENTER THE 4-SQUARES.
5, 13, 7, 15
CORRECT.  ENTER MORE 4-SQUARES OR 99
99
THE FINAL SOLUTION FOLLOWS:
```

			A
	B	B	A
	B	B	

variety of student errors. It is produced by calling the parts of the solution algorithm which check for 2-SQUARES and 4-SQUARES and analyzing the results.

Response A is provided since minterms 5 and 7, as well as forming a valid 2-SQUARE, are also part of a 4-SQUARE. This group of 4 (5, 7, 13, 15) is, of course, preferred. Response B results since no valid 2-SQUARE exists which contains minterms 5 and 15. Even though 9 and 13 form a valid 2-SQUARE, each of these minterms is also found in another 2 or 4-SQUARE. Hence, the algorithm correctly decides to defer selection of this pair as indicated by response C.

It is important to note that this type of detailed remedial feedback is automatically available regardless of the 3-variable or 4-variable Karnaugh Map problem being attempted and for any student error.

III. The MALT System

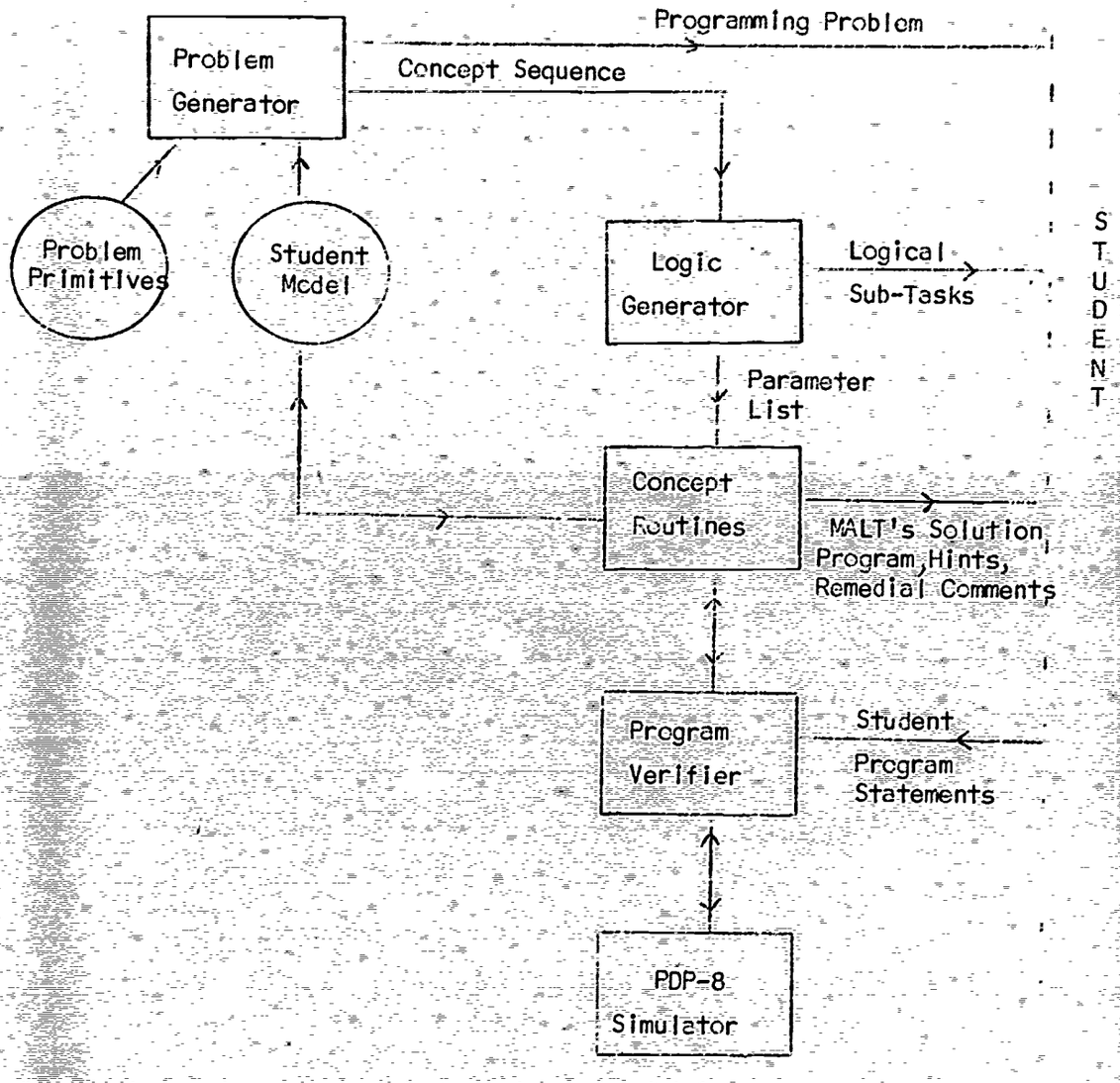
A. Overview

The subject matter of MALT is somewhat different from COMSEQ and CAILD in that it teaches machine language programming. However, MALT does utilize several of the concepts from COMSEQ in the problem solver mode to simulate the execution of a student program. These include the concepts dealing with octal arithmetic and logical operations.

In machine language programming, there are certain basic concepts which must be used over and over again in the design of a complete program. For example, pointers to data must be initialized, counters (to keep track of the number of loop iterations) must be initialized, masks must be set up, program loops must be terminated, data must be transferred into and out of memory, and accumulator and overflow status must be checked.

There are also somewhat more specialized concepts which often use these basic concepts as subroutines. For example, the concept which prints out the contents of a register uses the basic concepts concerned with transferring data into memory. Thirty-five concepts have been isolated as essential modules of machine language programs. As such, they can be combined to form a wide variety of rather lengthy and complex programs. This is the key factor which enabled the design of the MALT system.

The concept routines interact with the student during the design of his solution program. They solicit program statements from him and present remedial feedback if his response is incorrect. Figure 3 is a block diagram of the MALT system which shows the internal and external (STUDENT) flow of information.



MALT Block Diagram

Figure 3

The previous student performance determines what type of problem will be generated. This problem is presented to the student in natural language and also passed on to the system as a concept sequence and parameter list. Next, a list of logical sub tasks for each part of the problem is presented to the student. The system representation of this "flow-chart" is a sequence of calls to concept routines. These concept routines solve the programming problem and interact with the student to monitor his solution.

As the student undertakes each sub-task in the programming process, a corresponding Concept Routine is entered by the CAI system, which guides the student through the construction of that part of his program. During this phase, the student is constantly being given feedback as to the correctness of his program. If his program introduces logical errors, the system will point these out and offer helpful suggestions for their correction. If the system feels that the student might benefit from observing his program in operation, it also has the capability to simulate statement by statement program execution.

The system is constantly evaluating the student's performance and updating his permanent file. This is necessary because his achievement determines not only the difficulty of the problems given him, but also the amount of interaction which he receives during the design of his program.

Table 6 provides a complete example of the operation of MALT. The comments in square brackets have been added to clarify this example for anyone not familiar with the PDP-8 instruction set. Each of the 400_8 memory registers is 12 bits in length. The first octal digit of an instruction is the operation code. The last three octal digits specify the address of the

MALT In Operation
TABLE 6

```
Your problem is to write a program which will:
Read in 20 (octal) ASCII characters and store them in registers 232 through 252.
Form the sum of registers 232 thru 252 in the accumulator.
If this results in a non-zero LINK, stop with the (ACC)=7777, otherwise stop
   with (ACC)=0000.


Is this problem OK?
_yes

Here are the sub-tasks for the 1st line.
1) initialize a ptr to register 232
2) initialize a ctr with the value of -20(octal).
3) Read in a character.
4) Store it away using the ptr.
5) Update the ptr.
6) Update the ctr and if it is not zero, jump back to start of loop.

SUB-TASK 1:
Since we may need a ptr later, we will store the ptr value in memory
and move it to a register which we will use as the actual ptr.
Use M377 to store the ptr value, and M376 as the ptr.
M377:
 0232
Now, get the ptr value:
M000:
 1376 [Add in contents of M376]
WRONG!
"TAD 377" is coded as :1377
And deposit it into the ptr:
M001:
 3376 [deposit ACC into M376, clear ACC]
_

SUB-TASK 2:
We will store the ctr value in memory and then move it into another
register for use as the ctr.
Use register 375 to store the ctr value and register 374 as the ctr.
M375:
 7757
7757 can't be used as the ctr value.  Try again!
M375:
 7760  [-20 in 2's complement form]
First, get the ctr value:
M002:
 1375
And deposit it into the ctr.
M003:
 3374
_
```

```
SUB-TASK 3:
First, check the reader flag.
M004:
 5031
NO!
To check the reader flag is 6031
Next, jump back 1 register is it's not ready.
M005:
 5004 [JUMP to M004]
Now, operate the reader.
M006:
 6036

SUB-TASK 4:
M007:
 3376
Wrong!
"DCA I PTR" is :3776 [This should be an indirect deposit]

SUB-TASK 5:
Ok, update the ptr.
M010:
 2376

SUB-TASK 6:
Now, increment the ctr.
M011:
 2374
And if not zero, jump back to start of loop.
M012:
 5002
Wrong!
The loop starts in register 004 so it is :5004

Here are the sub-tasks for the 2nd line.
1) Initialize a ptr to register 232
2) Initialize a ctr with the value of -20(octal)
3) Add in the next register in the list.
4) Update the ctr and jump back if the ctr is not zero.

SUB-TASK 1:
Register 377 already contains the ptr value, 0232, so use it with M376 as the
ptr.
Now, get the ptr value:
M013:
 1377
And deposit it into the ptr:
M014:
 3376

Sub-Task 2:
We can re-use M375 as the ctr value and M374 as the ctr.
   M375: 7760
First, get the ctr value:
.
.
.
```

operand. If this is a number $\geq 400_8$, the instruction is an indirect address instruction. In this example, indirect addressing is used in conjunction with a pointer to enable an operation to be performed on a group of adjacent registers. A counter is used to control the number of times the operation is performed.

M377 stands for the memory register 377. The student's program starts at M000. The first character of each student response is underlined. The dialogue shown is that which would be received by a beginning student.

B. Problem and Logic Generation

Each programming problem can be thought of as consisting of three distinct phases; an input phase, a processing phase, and an output phase. There is a set of problem primitives associated with each. A problem primitive is a parameterized statement. $\mathcal{P} = \{I \cup P \cup O\}$ is the set of problem primitives where I,P,O represent the set of input, processing; and output primitives respectively and $\mathcal{P}$ is their union. The null primitive is also an element of I,P,O; consequently, many problems will have fewer than three phases.

There is a function d defined over the real numbers from 0 to 3.0 such that d(e) is the subset of $\mathcal{P}$ which may be used in problems for a student whose LEVEL is e. In addition there is a function f defined over $\{I \cup P\}$ such that $f(I_i)$ is the subset of $\{P\}$ which may follow input statement I_i in a meaningful problem and $f(P_j)$ is the corresponding subset of $\{O\}$ for P_j. The purpose of these functions is to insure that the difficulty of the programming problem generated is suited to the student's ability and that the problem "makes sense".

Problem generation, then, proceeds by first applying d to $\mathcal{P}$. An input statement, I_i, is selected at random from the set of eligible input primitives. Evaluating $f(I_i)$ further reduces the set of eligible processing primitives. One of these, P_j is selected at random and this process is repeated to select O_K. A final check is made to determine that the problem is sufficiently different from those already worked by this student and then the values of parameters in the selected primitives are randomly generated. Table 7 gives an example of some resulting problems in order of increasing difficulty.

TABLE 7

Sample Problems

Note: All randomly generated parameters are underlined

Level Range 0-1

1. Add the contents of register 150 to the contents of register 167.
2. Print out the message "HELLO".
3. Read in a series of ASC-II characters ending with a * and store them starting in location 120
4. Read in 31 ASC-II characters and store them starting at location 300. Search register 300 through 330 for the largest number.

Level Range 1-2

1. Read in a series of 3-digit numbers and store them starting at location 250. The input will end when the first character of a number is a "X".
2. Read in 24 (octal) four digit numbers and store them starting at location 242. Search registers 242 thru 265 for the 1st number which begins with the octal digits "70". (example 70XX)
3. Multiply the contents of register 211 by the contents of register 310.

Finally print out the 4-digit contents of the Accumulator.

Level Range 2-3

1. Search registers 160 thru 205 for the octal number 7215.

For registers 160 thru 205, print out the register number, 4 spaces, and the octal contents of that register.

2. Assume a table has been set up starting at location 120 consisting of a 2-character symbol followed by a number; there are 10 of these entries.

Search the table for the symbol "AN" and retrieve the corresponding number. If it is not in the table, then halt the program. Finally, print out the 4-digit contents of the Accumulator.

Associated with each problem primitive is a string of two digit numbers called the concept sequence. This sequence indicates which of the thirty-five basic concepts of machine language programming must be performed and in what order to program each primitive correctly. The average number of concepts needed per primitive is six .

Each primitive is programmed separately ; hence, the concatenation of the concept sequences for each of the three selected primitives is the concept sequence for the complete problem. This sequence specifies the form which the student's solution must follow and is equivalent to a flow-chart. It is interpreted by the LOGIC GENERATOR and printed as a list of sub-tasks prior to programming each primitive. Table 8 shows the concept sequence for the problem being programmed in Table 6 .

This restriction on the form of the student's program is essential in order for MALT to verify the correctness of the student's program and help him with the coding. Since the student is a neophyte in machine-language programming, it is felt that this imposed structure is useful in showing him how to attack a programming problem and outline its solution. He does have considerable freedom in coding each sub-task as will be discussed in the next section.

C. Program Coding and Verification

The coding of each sub-task is monitored by the concept routine in the sequence responsible for that sub-task. If the student's LEVEL exceeds the generation threshold for the routine, the coding for the associated sub-tasks (or sequence of sub-tasks) is provided by MALT. If the student's LEVEL is very low, he is led by the hand and program statements are requested one at a

TABLE 8

Concept Sequence

Concept Sequence: 23240715I03 232410 18

The "I" following concept 15 indicates this subtask should be performed indirectly.

The spaces indicate the end of a problem primitive

Concept Routine	Sub-Task
03	Terminate a loop
07	Input ASC-II Characters
10	Add a series of adjacent registers
15	Store the accumulator contents in memory
18	Check link status
23	Initialize pointers
24	Initialize counters

time. The intermediate student will normally enter a group of program statements at once.

Several conventions have been established to facilitate the generation of program segments and monitoring of student programs. All user programs begin in location 000 and all program constrants are placed at the top of memory beginning with location 377 and proceeding downwards. The middle areas of memory, locations 120 through 350, are reserved for lists and tables to be used by the student's program.

The existence of a program loop is assumed by the system whenever a pointer or counter is initialized. The physical start of the loop is the first memory register after the initialization process. By monitoring the beginning of a loop in this manner, the system can easily determine if the student has correctly designed his end-of-loop decision sequence. The most common programming mistake of this kind occurs when the student attempts to jump back to the initialization sequence instead of the main body of the loop.

Another program parameter which must be kept track of is the accumulator status. The simulated computer has neither a non-destructive deposit nor a destructive load instruction. Hence, the accumulator must be cleared prior to loading it with a given number and must be reloaded after a number has been deposited in memory, if the number is still needed. The status of the accumulator is going to determine which of several alternatives is to be pursued by certain of the concept routines in the design and checking of a program segment.

A complicating factor in determining accumulator status is the existence of logical branching or program jumps. The accumulator status may differ

depending on whether a concept routine was entered sequentially or through a program jump.

Forward jumps to yet unprogrammed concept routines also present a problem, as the memory location in which the new routine starts is not yet known. Consequently, MALT keeps track of the first memory location of each concept routine. If a jump is made to a previously programmed routine, the required instruction is provided immediately. In the case of forward jumps, a note is made of the memory location in which the jump instruction belongs and the concept routine to be reached. When this concept routine is finally programmed, MALT completes all prior jump instructions which reference it.

There are two important techniques used by the MALT system to judge the correctness of a student's program. The most common method is to analyze in detail each segment of the program as it is typed in, to determine if it performs the required functions. This is done on an instruction-by-instruction basis so that there is immediate feedback to the student.

Immediate verification implies that the system must have a detailed knowledge of the status of the user's program at all times. As the student formulates each response, the system also generates what it considers to be an appropriate answer. If the two do not match, the system must determine if other responses are possible. If so, the student's answer is compared with all such reasonable possibilities. When the system finally decides that the response supplied by the student is in error, it informs him as to the reason for this determination and supplies the best program alternative.

If the student's response matches any of those which the system generated, then it is accepted by the system as a valid alternative to its own solution.

Since this was not the expected result, however, the system must adjust its representation of the users program status to reflect the new conditions.

In the rare event that there are too many acceptable ways to program a particular sub-task, the program segment supplied by the student is simulated to determine its correctness. To verify the user's program through simulation, all conditions of the machine which might possibly affect final program results are determined. For example, if the program is intended to perform a particular operation depending upon the status of the overflow Link register, then only two initial states are necessary; the program is tested with a zero Link and again later with a non-zero Link.

Once the various initial states have been determined, the program segment can be simulated under each condition. The system decides, following each simulation, if normal program termination occurred. Conditions which might cause abnormal termination are such things as infinite loops, undefined instructions, or program branches which are directed outside the user's program segment. Any such conditions are corrected immediately by the student, the current set of initial conditions is re-established, and simulation is attempted again.

If any particular terminal condition indicates that the user's program did not perform its function correctly, MALT attempts remedial action. Since it is aware of the exact results which should have been obtained, it can provide a concise description of the error. It cannot, however, isolate the location of the error in the user's program. This determination must be left up to the student. However, the problem has been greatly simplified due to the system's diagnostics and the user's ability to observe his program in execution. If the student is unable to correct his program segment, MALT will generate a correct program segment for him.

IV. THE CAILD SYSTEM

A. Overview

The final system to be described is concerned with teaching students how to use integrated circuits (I.C.'s) to realize some of the "paper designs" problems encountered in COMSEQ. As many of the students enrolled in this course have had no prior exposure to digital or even analog electronics, CAILD performs an essential service. Figure 4 is a block diagram of CAILD.

The student first constructs his circuit off-line on a special student Logic Board. He has a fixed set of I.C.'s to work with which includes several types of NAND gates, inverters, four JK Flip-Flops, and four D Flip-Flops. The output of each gate can be monitored directly by CAILD. There are too many gate input points to justify giving CAILD the ability to monitor them all. Consequently, a test probe is provided which is connected to CAILD through an A/D converter. The student can move this probe to any test point in the circuit.

Once, the student has constructed his circuit, the logic board is connected to the general-purpose Input/Output buffer of the PDP-9 computer. The student describes his circuit by specifying each output equation and, in the case of sequential circuits, each Flip-Flop control equation in sum-of-product form. As each equation is entered, it is checked for syntax errors and all variables used are noted. Each equation is then converted to NAND format and stored in memory.

Debugging proceeds by testing the output of each equation for all possible combinations of the input and state variables. CAILD sets the input

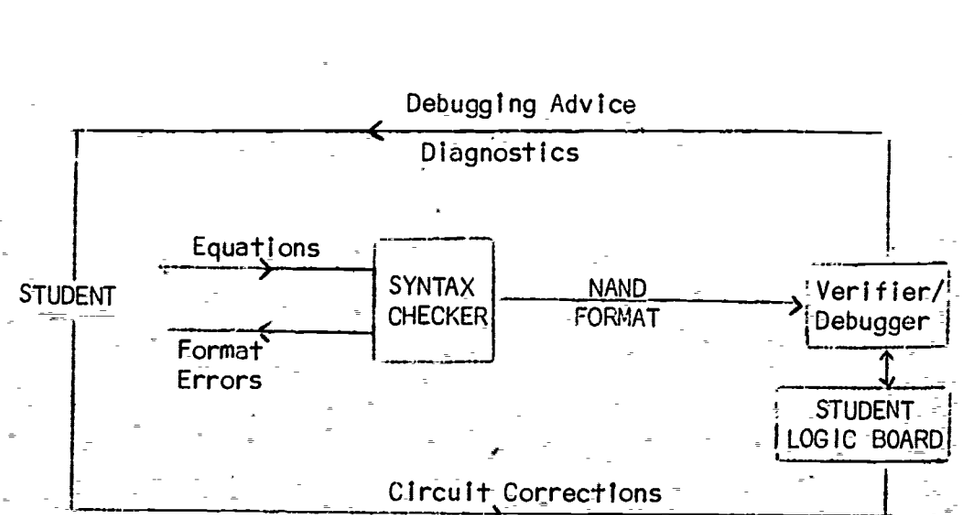


Figure 4

Block Diagram of CAILD

and state variables in the circuit to correspond to the combination being tested. The logic value present at the circuit point under test is compared to the value predicted by the equations. If they differ, CAILD aids the student in tracing down the circuit error and waits for him to correct it. CAILD then retests the equation.

When all equations appear error free, the circuit is free of any wiring errors or faulty components. In order to verify that the circuit actually does what the student intended, CAILD allows him to test it out. The student specifies a set of test conditions which are applied by CAILD to his circuit. The resulting output values and, in the case of sequential circuits, next-state values are displayed to the student.

CAILD can be used to test and debug a variety of digital circuits. As has been mentioned, there are two input, two output, and four state variables for sequential circuits of up to sixteen states. Combinational circuits with up to six inputs and sixteen output equations can also be designed. The additional input lines are obtained by allowing the Flip-Flop state variables to serve as input variables.

B. Error Detection

There are three categories of errors which may be detected.

1. Wiring error at input of I.C.
2. Improperly powered I.C.
3. Faulty I.C.

The presence of an error is indicated by a disagreement between the monitored logic value at a gate output poin. and the simulated or calculated value. To trace the error to its source, CAILD checks the logic value at the output of each lower level gate which is connected to the input of the gate in question. If any lower-level gate output is incorrect, then CAILD will switch its attention to this gate and its input points.

If all of the lower level outputs are correct, then there may be a wiring error. This is determined by monitoring the logic values at the corresponding input points of the gate in question. If there is a disagreement, then a wiring error exists. If there is no disagreement, then the gate in question is either faulty or powered incorrectly so CAILD will next check the voltage values at the power and ground points of this I.C. Figure 5 is a flow chart of the complete test procedure.

In all of this, the student is actively involved. He must refer carefully to his wiring diagram and inform CAILD of the location of all relevant gate output points. He must also physically move the test probe to all gate inputs which are to be checked and all suspicious I.C. power and ground points. Table 9 shows the interaction between CAILD and a student during the debugging process for the wiring diagram in Figure 6. The first error found is the absence of a ground connection to the inverter. The second error is a missing wire between M57 and M50. All student responses are followed by a ";".

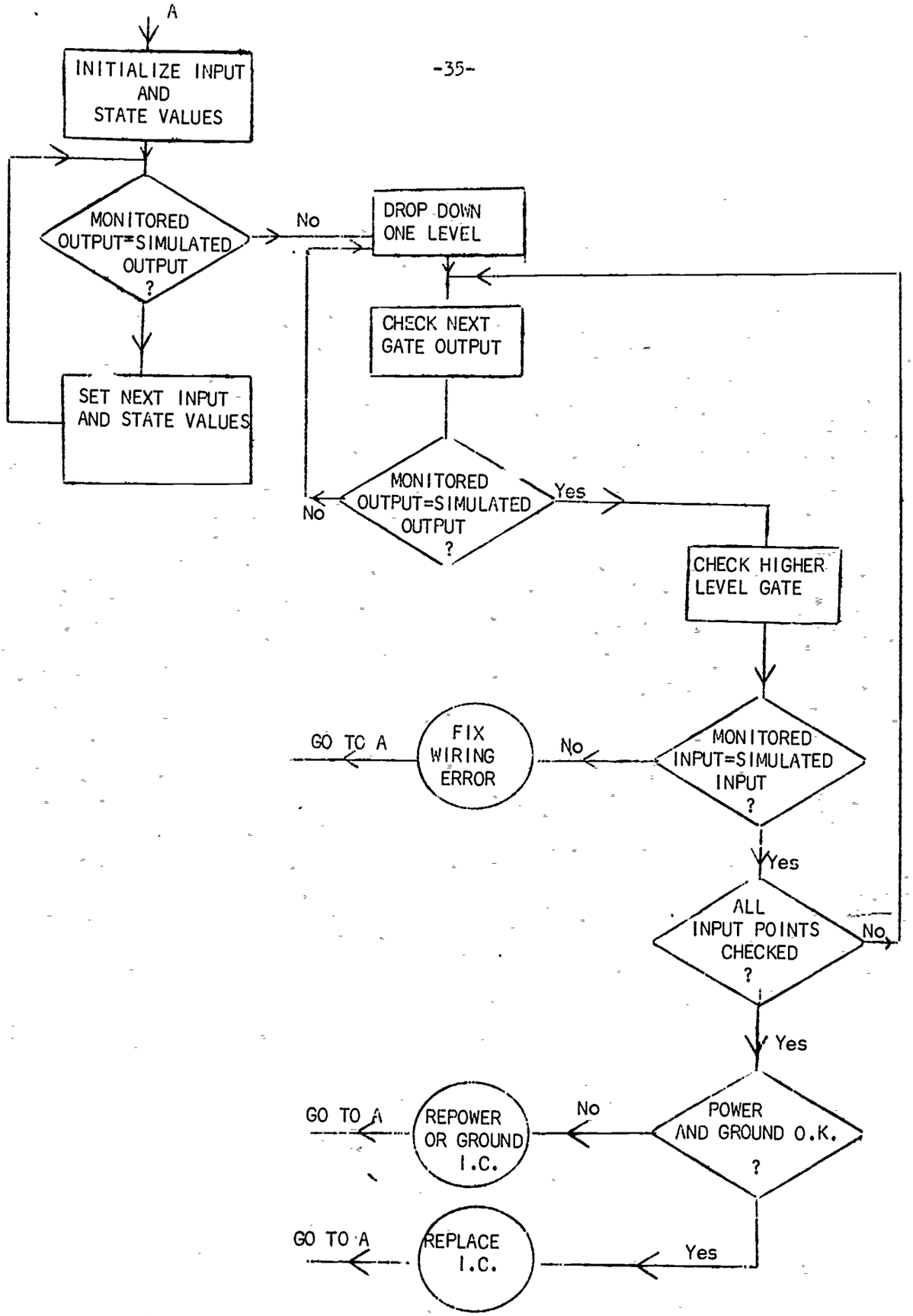


Figure 5 Flowchart for Debugging

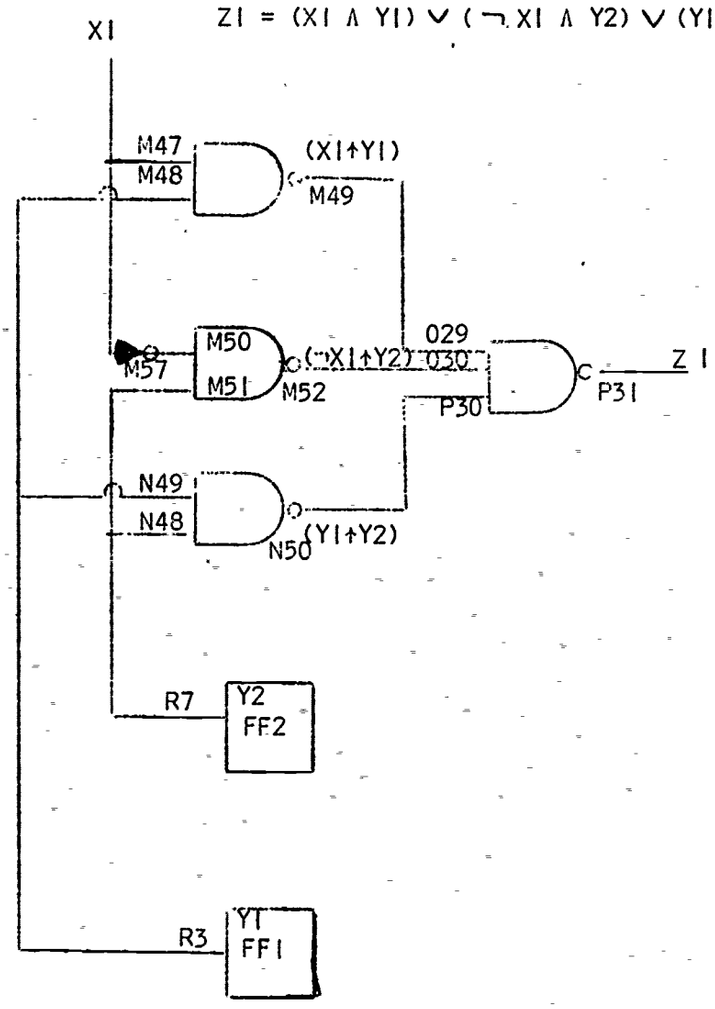


Figure 6

Equation Under Test

TABLE 9

CAILD In Operation

```
EQUATION UNDER TEST:
Z1=(X1↑Y1)↑(-X1↑Y2)↑(Y1↑Y2);
DO YOU WISH TO PRINT OUT ALL TEST CONDITIONS ?
(Y or N)
N;
SIMULATED OUTPUT CONDITION:        0
ACTUAL CIRCUIT OUTPUT CONDITION:   1

INPUT CONDITIONS:                  X1   Y1   -X1   Y2   Y1   Y2
                                    1    0     0    1    0    1

TELL ME THE OUTPUT CIRCUIT POINT FOR
THE GATE WHOSE OUTPUT IS Z1
P31;
TELL ME THE OUTPUT CIRCUIT POINT FOR
THE FOLLOWING TERM:
(X1↑Y1)
M49;
SIMULATED OUTPUT CONDITION:        1
ACTUAL CIRCUIT OUTPUT CONDITION:   1

PUT THE TEST PROBE ON THE INPUT OF NEXT HIGHER LEVEL GATE
WHICH IS CONNECTED TO THE OUTPUT OF (X1↑Y1)
TYPE ALT MODE WHEN YOU ARE READY.

THE WIRING SEEMS OK.
TELL ME THE OUTPUT CIRCUIT POINT FOR
THE FOLLOWING TERM:
(-X1↑Y2)
M52;
SIMULATED OUTPUT CONDITION:        1
ACTUAL CIRCUIT OUTPUT CONDITION:   0

PRESENTLY THE SYSTEM IS CHECKING ON THE VARIABLE -X1
TELL ME THE OUTPUT CIRCUIT POINT FOR THE INVERTER.
M57;
SIMULATED OUTPUT CONDITIONS:       0
ACTUAL CIRCUIT OUTPUT CONDITION:   1

PRESENTLY  THE SYSTEM IS CHECKING ON THE VARIABLE X1
SIMULATED OUTPUT CONDITION:        1
ACTUAL CIRCUIT OUTPUT CONDITION:   1
```

Table 9 (continued)

```
PUT THE TEST PROBE ON THE INPUT OF NEXT HIGHER LEVEL GATE
WHICH IS CONNECTED TO THE OUTPUT WE JUST CHECKED
TYPE ALT MODE WHEN YOU ARE READY.

THE WIRING SEEMS OK.
PUT THE TEST PROBE ON THE GROUND LEAD OF THE IC.
FOR JK F/F (7476).  THE GROUND IS AT PIN 13.
FOR ALL OTHER IC'S, THE GROUND IS AT PIN 7.
TYPE ALT MODE WHEN YOU ARE READY.

YOU HAVE EITHER AN OPEN CIRCUIT ON THE GROUND LEAD      [Note: MISSING GROUND
OR A WIRING ERROR TO THE GROUND LEAD.                  CONNECTION DETECTED]

CORRECT THE FAULT.  AFTER THE CORRECTION, TYPE ALT MODE
TO RETURN TO THE DEBUGGING PHASE.
TYPE ALT MODE WHEN YOU ARE READY.

EQUATION UNDER TEST:
Z1=(X1↑Y1)↑(-X1↑Y2)↑(Y1↑Y2);
DO YOU WISH TO PRINT OUT ALL TEST CONDITIONS ?
(Y OR N)
N;
SIMULATED OUTPUT CONDITION:        0
ACTUAL CIRCUIT OUTPUT CONDITIONS:  1

INPUT CONDITIONS:                  X1   Y1   -X1   Y2   Y1   Y2
                                    1    0    0    1    0    1

TELL ME THE OUTPUT CIRCUIT POINT FOR
THE GATE WHOSE OUTPUT IS Z1
P31;
TELL ME THE OUTPUT CIRCUIT POINT FOR
THE FOLLOWING TERM:
(X1↑Y1)
029;
THE CIRCUIT PT LABEL YOU PUT IN IS NOT A MONITORINT PT
ON THE CIRCUIT BOARD.  TRY AGAIN.
M49;
SIMULATED OUTPUT CONDITION:        1
ACTUAL CIRCUIT OUTPUT CONDITION:   1
```

Table 9 (continued)

```
PUT THE TEST PROBE ON THE INPUT OF NEXT HIGHER LEVEL GATE
WHICH IS CONNECTED TO THE OUTPUT OF (X1↑Y1)
TYPE ALT MODE WHEN YOU ARE READY.

THE WIRING SEEM OK.
TELL ME THE OUTPUT CIRCUIT POINT FOR
THE FOLLOWING TERM:
(-X1↑Y2)
M52;
SIMULATED OUTPUT CONDITION:          1
ACTUAL CIRCUIT OUTPUT CONDITION:     0

PRESENTLY THE SYSTEM IS CHECKING ON THE VARIABLE -X1
TELL ME THE OUTPUT CIRCUIT POINT FOR THE INVERTER.
M57;
SIMULATED OUTPUT CONDITION:          0
ACTUAL CIRCUIT OUTPUT CONDITION:     0

PUT THE TEST PROBE ON THE INPUT OF NEXT HIGHER LEVEL GATE
WHICH IS CONNECTED TO THE OUTPUT WE JUST CHECKED
TYPE ALT MODE WHEN YOU ARE READY.

EITHER YOU HAVE A WIRING ERROR OR AN OPEN CIRCUIT  [Note: MISSING WIRE FROM
AT THE TEST POINT.                                        M57 to M50]

CORRECT THE FAULT.  AFTER THE CORRECTION, TYPE ALT MODE
TO RETURN TO THE DEBUGGING PHASE.
TYPE ALT MODE WHEN YOU ARE READY.

EQUATION UNDER TEST:
Z1=(X1↑Y1)↑(-X1↑Y2)↑(Y1↑Y2);
DO YOU WISH TO PRINT OUT ALL TEST CONDITIONS ?
(Y OR N)
N;
DEBUGGING COMPLETED FOR ONE EQUATION.
DEBUGGING PHASE COMPLETED.
DO YOU WISH TO TEST YOUR CIRCUIT? (Y OR N)        [Note: NO ERRORS
N;                                                       REMAINING]
OPERATION SUCCEEDED!!!
```

V. Evaluation and Conclusions

The purpose of these CAI systems is to augment classroom instruction rather than replace it. None of them is really intended to be an end in itself, but their purpose is to help the student bridge the gap between theory and independent application. Experience has shown that these CAI systems are far more effective than conventional homework assignments for this purpose.

After using MALT for approximately two weeks, students are required to program and run a sizeable problem in assembler language. Students who have acquired a feel for machine language programming through interaction with MALT find it easier to strike out on their own and complete their assembler language program. CAILD has not yet been evaluated; however, the authors anticipate that the experience gained by students in debugging digital circuits under the guidance of the CAILD system will prove beneficial to them in later laboratory projects.

Since each of these systems is "intelligent" in its small domain of application, they are very flexible and are not limited in the problems they can solve. For example, COMSEQ is available as a problem-solving assistant to help advanced students design and simplify the logic circuits to be debugged and tested by CAILD. In fact, the student can use COMSEQ to derive his circuit equations in minimal Sum-of-Product form.

CAILD itself can be used to build and debug digital circuits such as counters, adders, decoders, and shift registers. The only constraint is that the number of independent circuit variables not exceed six (two inputs and four state variables or six input variables). Combinational circuits can have

up to sixteen output lines; sequential circuits can have up to sixteen states.

Present plans call for adding a "front end" to CAILD which will allow a student to enter a state table for a problem or select one of a variety of circuits to build. CAILD will then aid him in determining a correct set of equations for this circuit. Finally, CAILD will teach him how to wire the individual I.C.'s prior to his constructing the circuit.

Due to its modular construction, certain of the concept routines in COMSEQ are used as subroutines by more advanced concepts in COMSEQ and also by MALT (during program simulation). For example, a student who has produced the state table for a counter can, if he wishes, continue on and derive a truth table for the Flip-Flop control lines, and also determine the minimal combinational logic required.

It is relatively easy to add new concepts to COMSEQ. A new problem generator and solution routine must be written. There is an instructor-mode available which can be used to make the necessary additions to the concept tree. Similarly, new problem primitives can easily be added to MALT. The concept sequence for this primitive must be defined in terms of the thirty-five basic machine language programming concepts which are available for use in MALT. Also, the other problem primitives it can be interfaced with must be identified along with the LEVEL range in which it may be used.

COMSEQ and MALT also attempt to individualize the problem selected for the student and the depth and pace of instruction provided. COMSEQ, in particular, learns about the student as the course progresses and uses its expanded knowledge of the student to decide how quickly he should progress from plateau to plateau in the concept tree and which one of the concepts within a plateau should be studied next.

The authors feel that this set of generative CAI systems will continue to be useful in augmenting the learning tools available for a beginning student in computer science and electrical engineering. Hopefully, the concepts employed in developing these systems can be applied successfully in other course areas. Current research includes the design of a system patterned after COMSEQ for teaching problem-solving in high school algebra. An author language for facilitating the design of future CAI systems is also being implemented.

A questionnaire was distributed to students using COMSEQ and MALT this past semester (Spring 1973). A summary of student responses to some of the questions in this questionnaire is given in Table 10. These results clearly indicate student acceptance of the role fulfilled by these systems in the course.

TABLE 10

Student Evaluation

Note: Questions 1-8 refer to MALT only; 10-17 to COMSEQ only

For questions 1-15 the numbers of students giving the following responses are tabulated

	Strongly Disagree	Disagree	Uncertain	Agree	Strongly Agree
1. The system was useful in introducing me to machine language programming.	0	2	1	18	12
2. It was relatively easy to learn to use the batch version of the assembler since I had been introduced to programming concepts through MALT.	0	5	4	15	7
3. Since the subtasks were always laid out for me, I felt very constrained using MALT.	0	19	9	5	0
4. Because the subtasks were laid out, I only learned the mechanics of programming and didn't really understand what was going on.	1	16	8	5	2
5. The approach taken in printing out the subtasks was good as it taught me how to organize a machine-language program.	0	2	7	20	4
6. The problem became more difficult as my level increased.	1	3	7	19	3
7. There was a good variety in the problems I received in MALT.	1	12	6	13	
8. In general, I enjoyed the interaction with MALT.	0	3	6	21	3
9. In general, I preferred the use of CAI in this course to conventional homework (both CAI systems)	0	2	4	11	16
10. I was concerned that I might not be understanding the material.	3	13	4	7	1

11. CAI is an inefficient use of the student's time.

11	12	2	2	2

12. CAI made it possible for me to learn quickly.

1	6	7	15	0

13. The CAI System does a good job of selecting concepts.

6	6	8	8	1

14. In view of the effort I put into it, I was satisfied with what I learned while taking CAI.

2	1	0	17	8

15. Overall, I enjoyed working with CAI. (CONSEQ)

0	2	2	23	2

For questions 16-17 the numbers of students giving the following responses are tabulated

All of the time	Most of the time	Some of the time	Only Occasionally	Never

16. I found myself just trying to get through the material rather than trying to learn.

1	5	5	13	4

17. I was given answers but still did not understand the questions.

0	1	13	5	10

VI. Problem Generation And Solution In High-School Algebra

A. Overview

This chapter will describe a generative computer-assisted instruction (CAI) system for high-school algebra. The classes of problems which the system can handle are word problems, manipulation problems, and graphing problems. Word problems test a student's reasoning and problem-solving ability; manipulation problems test his grasp of the fundamental skills and techniques; graphing problems test his basic understanding of graphic methods and ability to plot equations.

Problem generation for all three types is based on a probabilistic grammar. A table of probabilities covering each rule in the grammar is used to tailor the problem to the individual student in terms of desired emphasis and level of difficulty. For example, the table might favor generation of an "age" problem if additive concepts were to be emphasized; whereas, a "percentage" problem would be more likely if multiplicative concepts were to be emphasized. If a difficult problem is desired, the probabilities associated with complex problem primitives or complex expressions would dominate.

Like problem generation, solution monitoring is dependent on a student's past history. When a student begins a new concept he will be led step by step through the problem and essentially "learn by example". As his proficiency increases, the system will require less and less interaction and provide less instruction. Solution monitoring (after the student is sufficiently advanced) consists of checking to see that his approach appears reasonable, accurate, and error free. If the student appears to be having trouble or if he asks for help, parts of the solution will be given.

B. Manipulation Problems

Manipulation problems are used to increase the students skill in algebraic operations. For example, if 4X+3=7, what does X equal? The intention is to supply a different type of manipulation problem for different concepts to be emphasized. In order to obtain different types of problems from one generation system, a prabablilistic grammar is used to generate each expression of the equation. See page 13 for an explanation of probabilistic grammar(s). The grammar is implemented as a set of recurrsive PL/I functions. A table of probabilities is used in the call to the generator routines to supply initial probabilities for each rewrite rule. Also in the table is a set of decrement values which are used to establish a linear conditional probabilities effect. The currently used grammar is shown in Table .

In order to generate problems of a specific type, the appropriate probabilities table need only be passed to the generating routines. An interactive program is available to the course author so that he can experiment with the probabilities table for each concept before incorporating it into the concept itself.

As an example, if the derived problem type is a quadratic equation in one variable, the probability and decrement for a following term are set to values which tend to produce three or more terms in the final expression. Each <term> is rewritten as <factor>. The decrement values are applied between each call to <term> . Hence, the probabilities asscciated with each re-write rule of <factor> are different for each term of the expression. That is, the probabilities are adjusted such that in the first term the re-write rule <factor>→<VAR>↑<POWER> is chosen; in the second term, <factor>→<CONST VAR>;

Grammar for Generating Polynomials

TABLE II

1. EXP→TERM {+ TERM}|{-TERM}
2. TERM→FACTOR {* FACTOR}|{/FACTOR}
3. FACTOR→(EXP)↑N|(EXP)|VAR|CONST VAR|VAR↑N|CONST
4. VAR X|Y|Z
5. CONST→ 0|1|...|9|CONST CONST
6. N →1|2|3

There is a probability associated with each re-write rule.

KEY: Rewrite Rule 1 states that an expression can be replaced by a single term followed optionally by a plus or minus sign and another term. (Any quantity enclosed in brackets is optional).

Word problems are used to increase the students problem solving abilities in the sense of converting a verbal description of the problem into equations. As in the case of manipulation problems, the word problem generated should be appropriate to the concepts to be emphasized. An additional difficulty in generating word problems is that the solution must be generated as part of the problem generation procedure. For this reason, even though the word problem generation routines can be modeled as a probabilistic grammar, they are actually implemented as a set of PL/I routines which manipulate a LISP-like data structure. The data structure (actually there is a separate data structure for each problem type due to storage limitations)contains a set of problem primitives which are strung together in order to construct a problem. The data structure imposes a difficulty ordering on the primitives, associates a solution primitive with each problem primitive and additionally indicates which primitive can reasonably follow from another in order to make a meaningful problem. As with manipulation problems there is a table of probabilities associated with each problem type to be generated which controls the manner in which the data structure is traversed. It also controls the probability of

Appendix

Abstract Problems

An abstract problem is a triple (data, unknown, relation) where data is a vector of input variables. The unknown is the variable whose value is sought as the solution. The relation defines a function which assigns a unique value to the unknown for each vector of data input values. An abstract problem can be represented by a function whose domain corresponds to data, whose range corresponds to the unknown, and whose rule is given by the relation.

An interpretation I of an abstract problem consists of an association of an object(s) to the abstract problem and the assignment of specific attributes to the data variables and the unknown.

An abstract problem can have many interpretations. The values of the attributes of the interpretation must belong to the domain of the problem.

Let Pj(I) denote the problem Pj with the interpretation I; unknown (Pj,I) denotes the solution to the problem Pj(I). Let the class of problems generated (or represented) by Pi using all allowable interpretations be denoted Ci.

Pi is a subabstract problem of Pj, denoted $Pi \leq Pj$, if $Ci \leq Cj$.

Two abstract problems, Pi and Pj, are weakly equivalent if $Ci = Cj$.

Let Pi and Pj be abstract problems. Denote their domains by Di and Dj,

selecting a more difficult problem primitive. This table is updated as a student progresses in order to prevent too many similar problems from appearing.

A grammar for a simple age problem and its structure representation are given in Table 12 and Figure 7. The grammar shown is only capable of generating a few sentences; the grammar actually used is considerably more complex. The answer primitives shown are in a simple form which can be used to extract the solution to the problem (in the sense of writing the equations) but they currently do not supply enough information to allow the program to lead the student from word problem to equations. They will supply this information eventually but the final form is not set at this time.

One could look at each rewrite rule of the grammar as a function which returns its value in a manner similar to the manipulation generation routines. The reason for not doing so are technical characteristics of PL/I involving effective storage utilization and ease of modification. If one looks at the rewrite rules as functions which return values, the possibility of primitives which are evaluated for their effect rather than their value comes to mind. For example, placing a picture on the screen to supplement a geometric-type word problem would involve an entity which causes the figure to be displayed. The intention is to experiment with this later.

Given the above data structure, word problem generation is relatively straight forward. The data structure is scanned using the probabilities table to determine the relative difficulty of the problem, substitutions are made for variables in the primitives (for instance, C1 and C2 in Table 12), and the solution equation(s) is generated for the problem from the answer primitives.

D. Graphing Problem Generation

The generation methods for graphing problems in a high school environment require little sophistication. Most problems are concerned with finding the slope, X-Y intercept, or perhaps the points of intersection of two first or second degree curves on a plane. The expression generator for the manipulation problems can be used to generate the equations to be graphed. A simple probabilistic grammar can be used to decide the type of problem and the solution method. In short a combination of techniques from manipulation and word problem generation will be used. Pure graphing problems in this system will be a series of short, fast interactions between machine and student in which the student receives immediate feedback after any mistakes.

Graphing is a problem solving method and not an end in itself and the intention is to have graphics as available to the student for problem solving as algebraic manipulation. After the student has progressed sufficiently, graphing may be the required solution method for word problems (particularly for problems with several equations and several unknowns).For the final stages of a student's CAI work, he should be able to decide when to use graphics as the solution method for a problem.

Several general remarks about problem generation need to be made. In all three cases, the probability tables passed to the generating routines can control problem difficulty as well as problem type. In all three cases, an instructor experimentation mode is, available so that the instructor can experiment with the probability tables and data structures to test the output of the generation routines. In the case of word problems, an additional difficulty can occur, particularly with a slow student. The number of possible problems is much smaller than for the other two types and for some concepts there may be only two or three possible first

sentences. At the very least, there should be some way to insure that the number of repetitions is minimal. (A problem is certainly easier if it can be solved by direct analogy with a previous problem.) Therefore, information concerning previous word problems will be associated with the student's record. The exact format is not set as yet.

E. Solution Monitoring

In a student's early stages of use, the system will lead him step by step through the solution. As a student progresses through the course his skills increase to the point where he no longer desires nor needs to be led through the problem. It is, of course, much more difficult to monitor his work at this stage. The system can check that his work is algebraically correct, start the problem over if the student desires, or in many cases suggest how to proceed to a solution. It is difficult in complicated problems to keep the student from going off on a tangent. To circumvent this problem to some extent, a desk calculator mode for both manipulation and graphing is planned which is partially coded and debugged.

This mode is envisioned as being available to the student from any point to allow experimentation (without machine supervision) in a sort of scratch paper manner. For manipulation problems, commands to perform common manipulations will be available (e.g. 'ADD 3X' will add 3X to an expression). In the graphics mode, provision for plotting simple equations and connecting points under student control will be made, as well as allowing the student to change the scale of his coordinate system.

This mode is considered to be important in developing a student's skills

with the problem solving tools of algebraic manipulation and graphing without too much computer interference. The most difficult part of solution monitoring is, of course, translation from a written problem description to the equations necessary to solve the problem. This process is only partially understood in humans and is highly heuristic (Paige, 1966). For this reason there is not much the system can do for the student except to lead him through step by step in a very rigid manner. In the case of a more advanced student, the system can tell him whether his equations are right or wrong and, perhaps, show him one way to derive the correct equations. At this point, it is not clear how much can be done in this area and only actual experimentation will resolve this question.

Grammar for Algebra Word Problems

TABLE 12

PROB= P1 P1X|P2 P2X

P1X = P11 P11X|P12 P12X

P11X = P111|P112

P12X = P121|P122

P2X = P21 P21X|P22 P22X

P21X = P211|P212

P22X = P221|P222

P1 = 'JACK IS C5 YEARS YOUNGER THAN HIS FATHER.'

P2 = 'JACK'S AGE PLUS HIS FATHER'S AGE IS C3 YEARS.'

P11 = 'THE SUM OF THEIR AGES IS C3.'

P12 = 'THE PRODUCT OF THEIR AGES IS C4.'

P111 = P121 = 'HOW OLD IS JACK?'

P21 = 'THE DIFFERENCE OF THEIR AGES IS C5'

P22 = P12

P211 = P111 = P221

P212 = P222 = 'HOW OLD ARE JACK AND HIS FATHER?'

C1 = 5|6|...|20

C2 = C1 + 18|19 ...|30

C4 = C1*C2

C5 = C2-C1

C3 = C1+C2

A1 = 'C5=C1-C1'

A2 = 'C1+C2=C3'

A11 = 'C1+C2=C3'

A12 = 'C1*C2=C4'

A121 = A111 = '?C1'

A21 = A1

A22 = A12

A211 = A111 = P221

A212 = A222 = '?C1?C2'

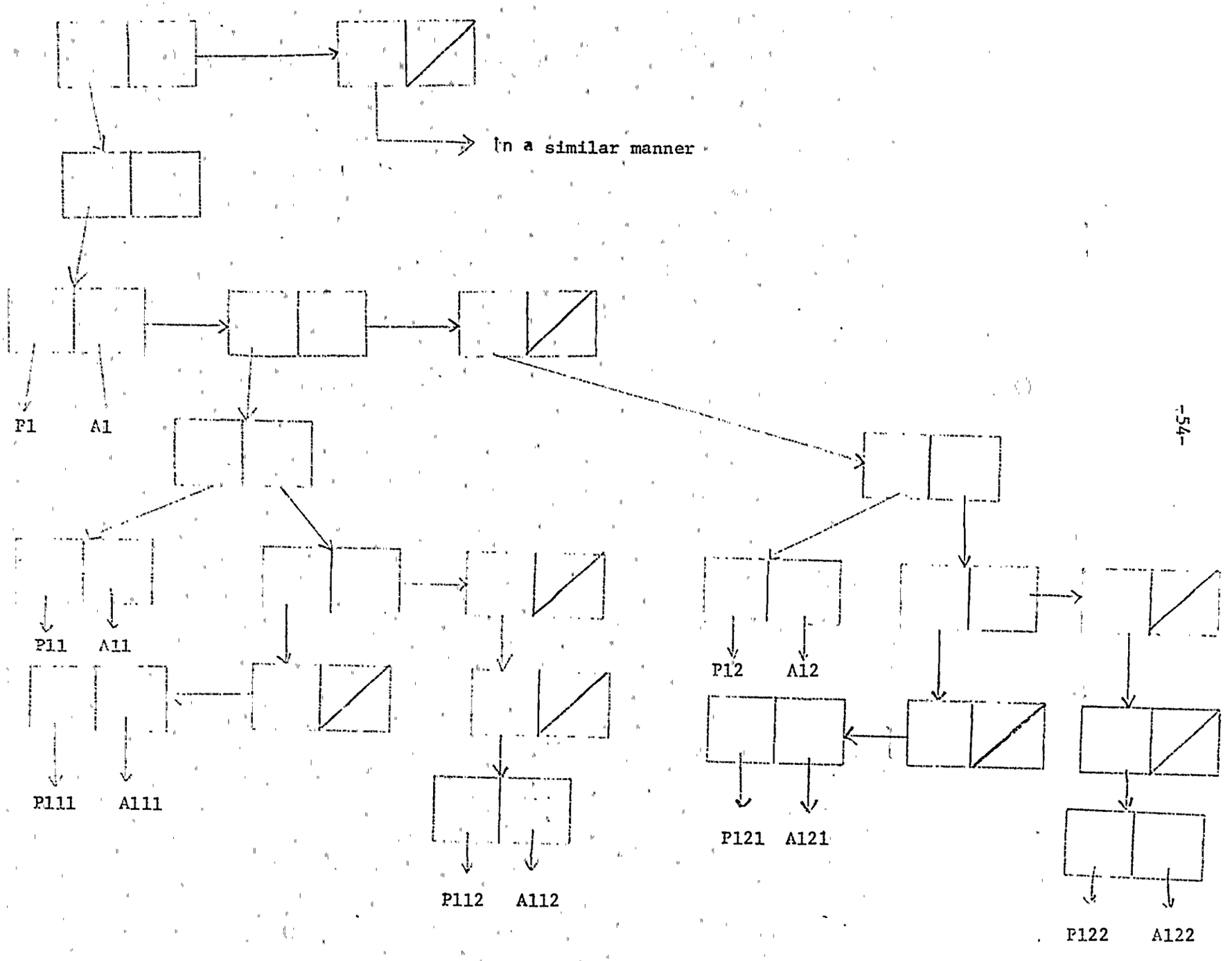


Figure 7

Part 2 structure flow chart problems

VII. Formal Model of Problem Generation and Solution

A. Introduction

Problem generation is described as originating from a semantic network (Quillian, 1969). This process is an extension of Carbonell's work to quantitative problem solving (Carbonell, 1970) and is motivated by Polya's classic work on problems (Polya, 1945). Indeed, this research can be thought of as a step in the automation of Polya's heuristics. Problem solution is closely related to problem generation. The plan for problem solution is generated at the same time that the problem is generated.

In the above systems four basic elements can be discerned, memory, reasoning, input and output. Memory contains factual information such as course material and the student model. Reasoning usually takes the form of algorithms. It generates problems, solves problems, and makes decisions on the basis of input. If the reasoning component has learning capabilities it can modify the contents of memory. These four elements are related according to the Block Diagram in Figure 8.

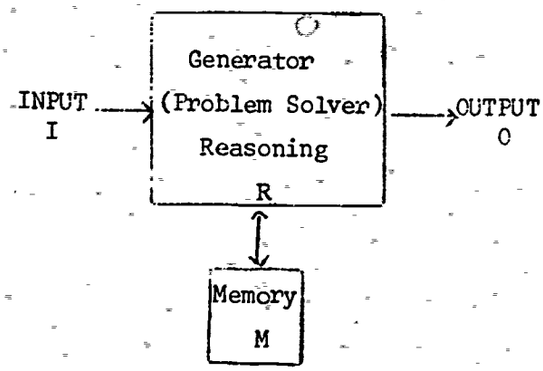


Figure 8

Block Diagram of Problem Generator/Solver

The form that each of these elements can be given is investigated in following sections.

B. Memory and Problem Generation

A general view of problems is taken. Let P be the class of all problems under consideration.

Any problem can be analyzed into three parts--the unknown, the data, and the condition (Polya, 1945). P is partitioned into subclasses by taking two problems to be equivalent if they are represented by the same triple (data, unknown, relation); i.e., same without values and interpretation of variables.

The class of triples can also be grouped into equivalence classes according to problem type. Two expressions for a relation are equivalent if they represent the same relation. The classification of problems can be carried further by classifying relations according to various properties.

The significance of these problem equivalence classes is that all the problems in a class can be represented by their common triple. Moreover, the system can select the unknown from among data items. Hence, problem equivalence classes can be represented by the pair (data, relation). The data itself can be represented by (object, attribute, value) triples. Since the representations will not have values the final representation is (object(s), attribute(s), relation).

The classifications are summarized by the scheme, problem → (data, unknown, relation) → (data, relation) → (object(s), attribute(s), relation) where this analysis proceeds from a specific instance of a problem to the most general representation of the problem class. For generation these processes are reversed. A brief introduction to a formal theory of problem classification, generation and solution is presented in the Appendix.

M represents the system's knowledge of the subject matter (at least a large part of M does) and can conveniently take the form of a semantic network. A semantic netowrk is a graph structure with items corresponding

to the nodes and relationships to the arcs. The items can be objects, attributes, or pointers. Values are associated with the attributes. The relationships are semantic relationships if the semantic network is applied to natural language or logical relationships of interest to a particular subject matter. Carbonell and Wexler have demonstrated the utility of semantic networks for storing factual information. Simmons (1970) has shown their utility for natural language analysis and generation. Melton (1971) has studied the automatic generation of semantic nets from text. Schwarcz (1970) has applied semantic nets to deductive question answering. Quillian (1969) has applied them to the comprehension of English text. By extending a semantic net and basing it on a 4 tuple (object, attribute, value, relation), a semantic network can also be used for problem solving.

Semantic memories, then, can be used for several purposes, including storage of subject dependent information, natural language analysis and generation, problem generation and solution.

Problem generation with regard to a pair (data, relation) consists of 1) selection of an interpretation, 2) selection of an unknown, 3) generation of values for data attributes, (Steps 1 and 3 are an interpretation using the semantic network as the domain of interpretation) 4) selection of a particular relation if a class of relations is specified in the pair. This generation process is the reverse of the prior problem classification.

In terms of the model problem generation is performed by the operator R. R generates a problem as a function of memory M (the pairs (data, relation) and the student's level).

In addition, if a semantic memory is used to represent the subject information, the system is also able to generate factual questions, even for a subject which is oriented to problem solving.

C. Problem Solution

In the model, R also corresponds to the problem solution mechanism.

A pair (data, relation) is called an abstract problem and corresponds to a concept(s) in the concept tree. For solution, abstract problems or groups of abstract problems can have associated solution methods.

A problem solving system is described by a pair (P,S) where P is a collection of abstract problems and S is a solution method for solving interpretations of P. For the problems of a particular subject area, suppose $P = P_1 \cup P_2 \cup \ldots \cup P_n$. Since no one solution method may be practical for P, each Pi has an associated Si.

A problem solving system needs some sort of planning mechanism with which to approach a problem. A plan is defined as some composition of Si's. For a system which solves a restrictive class of problems the plan need be no more than a fixed sequence of steps to be carried out. However, for a large class of problems the system will need plans that vary from complex problem to complex problem. A plan will indicate a decomposition of a complex problem into basic subproblems which belong to the Pi. Each subproblem has an associated solution method Si. A corresponding composition of the solution methods will yield a solution method for the complex problem.

Thus, the class of problems generated by abstract problems {Pi} is extended while keeping the same Si's. Various constructions are applied to the basis {Pi}, yielding complex problems not in any of the classes Pi. Hopefully, the new problems will be solvable using the Si's combined according to the type of construction used. Hence, if $(P_1 \cup P_2 \cup \ldots \cup P_n, \{Si\})$ and $P \supset P_1 \cup P_2 \cup \ldots \cup P_n$ then $(P, \{Si\})$, where Pi are the representations within the computer and $\supset$ is the larger class of problems solvable by Si.

Another way of extending a class is by transformation, i.e. suppose P is solvable by S and $T: P' \rightarrow P$ is a Transformation of P' to P. Under certain conditions P' is also solvable by S. These extension techniques are generation mechanisms and also possible strategies for a planning mechanism.

In a generative system the plan for problem solution can be a product of problem generation. The system constructs the problem out of basic building blocks; i.e., abstract problems, each of which has an associated solution method. A corresponding construction applied to the solution methods comprises the plan for solving the constructed problem.

Indeed, the implementation of the plan as a set of procedures for solving the problem is a special case of automatic program synthesis (Manna, 1970). The solution method is synthesized out the Si's. The solution procedure's control structure, expressed by the plan, will correspond to the particular construction used to compose the problem out of the subproblems Pi.

A flow diagram which summarizes the problem generation and solution process is shown in Figure 9.

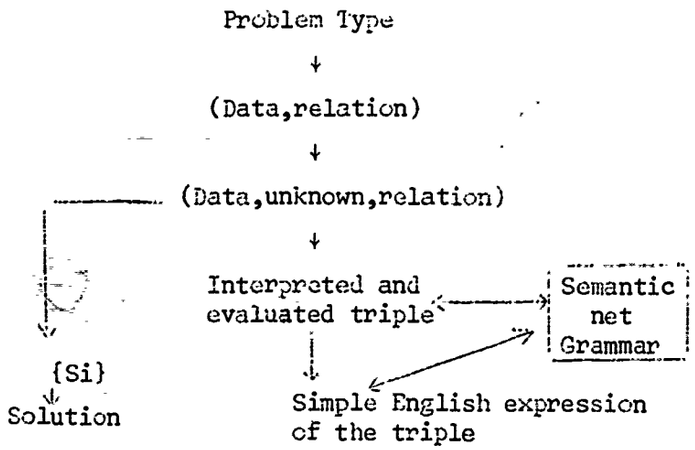


Figure 9
Flow Diagram of Generation/Solution Process

D. Input and Output

Semantic networks can also express semantic relationships of objects. In some applications, English has been mapped into a semantic net and related information retrieved (Carbonell, 1970, Schwarcz, 1970). Semantic nets have likewise been used for English generation.

Thus, a semantic net can be used to convert an interpreted abstract problem to simple English statements--perhaps one for data, one for the unknown, one for the relation.

For quantitative responses from the student, the system does not need sophisticated input conversion to tuples. Non-quantitative answers, however, would need conversion to the intermediate form in order to make use of the semantic net. Such conversion is also necessary for the system to function as a problem solver and question answerer which accepts English problems as input rather than generating them.

E. Learning

Learning by an intelligent system for CAI includes updating the model of the student, acquisition of new information, or revision of old information contained in memory.

Learning can be rote learning or generalized learning (Samuel, 1967). Further, learning implies the acquisition of new knowledge. The new knowledge can be received by the system from the designer (its need having been indicated by the system), obtained by the designer and the system, or acquired by the system itself.

Learning, then, can be realized on several levels. Level 0 is not learning but is included for completeness. Level 0 occurs when the system designer inputs information or solution methods into the system. Level 1 occurs when the system has received a question or problem or generates one which it is unable to solve. The system remembers these and informs the designer who updates the semantic memory or increases the solving power of the system. Levels 0 and 1 are rote learning. Level 2 learning consists of solving a new type of problem by discovery (or selection) of similarities or analogies with known problem classes. Level 3 learning consists of solving a new type of problem by analyzing the problem to discover its structure and then formulating a solution plan using the available solution methods. Level 2 and level 3 are generalized learning. All of these types of learning result in an increase in the question answering or problem solving power of a system.

Another type of learning of concern for CAI is that of the student user. The CAI system should increase not only its own state of knowledge, but also that of the student user.

Learning would come directly into play for a generative system if the process of generation were reversed and the system had to analyze a new problem to discover its corresponding synthesis and solution.

References

1. Blount, S., and Koffman, E. B., 1973, "Teaching Coding Through Generative CAI". Proceedings of the 1973 British Computer Society Symposium, DATAFAIR, April, Nottingham.

2. Booth, T. L., 1971, Digital Networks and Computer Systems, Wiley and Sons, New York.

3. Carbonell, J. R. 1970, "Mixed-Initiative Man-Computer Instructional Dialogues," Bolt Beranek and Newman, Inc. BBN Report.

4. Koffman, E. B., 1972, "A Generative CAI Tutor for Computer Science Concepts," Proceedings of the 1972 Spring Joint Computer Conference, AFIPS, pp. 379-389.

5. Manna, Z. and Waldinger, R. J., 1970, "On Program Synthesis and Program Verification," Artificial Intelligence Group, Technical Note 52, Stanford Research Institute, Menlo Park, California

6. Melton, T. R., 1971, "IT1: An Interpretive Tutor", Technical Report No. NL-4, Dept. of Computer Sciences and Computer-Assisted Instruction Laboratory, The University of Texas, Austin, Texas.

7. Paige, J. M. and Simon, H. A., 1966, "Cognitive Processes in Solving Algebra Word Problems," in Klienmunty, Benjamin (ed) Problem Solving: Research, Method, and Theory, John Wiley and Sons, N. Y.

8. Polya, G., 1945, How to Solve It, Doubleday and Company, Inc., Garden City, New York, 1945 and 1957.

9. Quillian, M. R., 1969, "The Teachable Language Comprehender: A Simulation Program and Theory of Language," Communications of the ACM, Vol. 12, No. 8, August, pp. 459-475.

10. Samuel, A., 1967, "Some Studies in Machine Learning Using the Game of Checkers. II. - Recent Progress," IBM J. R. and D 11, 6, pp. 601-617.

11. Schwarcz, R. M., Burger, J. F., and Simmons, R. F., 1970, "A Deductive Question-Answerer for Natural Language Inference," Communications of the ACM, Vol. 13, No. 3, pp. 167-183.

12. Simmons, R. F. and Slocum, J., 1970, "Natural Language Research for Computer-Assisted Instruction," Computer Sciences and Computer-Assisted Instruction Laboratory, The University of Texas, Austin, Texas.

13. Wei, M., 1973, "CAILD: A CAI System for Debugging and Testing Digital Circuits," Master Thesis, The University of Connecticut, Storrs, Conn.

">" is a partial order on abstract problems. Pi is strongly equivalent to Pj if and only if Pi > Pj and Pj > Pi. Also, if Pi > Pj then Pj - Pi. ">" can be interpreted as more difficult than

Various construction techniques for creating new abstract problems from given problems can be defined.

Abstract problems and their relations have significance for problem generation. It is possible to introduce solution methods into the discussion by attaching a predicate to an abstract problem. And, then, questions of significance of constructions on abstract problems for solution methods can be studied (Perry, 1973).